Physically based hydraulic erosion simulation on graphics processing unit

Conference Paper · January 2007

DOI: 10.1145/1321261.1321308 · Source: DBLP

3 authors, including:

Alexei Sourin
Nanyang Technological University
142 PUBLICATIONS **1,402** CITATIONS

SEE PROFILE

Physically Based Hydraulic Erosion Simulation on Graphics Processing Unit

Nguyen Hoang Anh*
Nanyang Technological University

Alexei Sourin†
Nanyang Technological University

Parimal Aswani‡
TQ Global

Abstract

Visual simulation of natural erosion on terrains has always been a fascinating research topic in the field of computer graphics. While there are many algorithms already developed to improve the visual quality of terrain, the recent simulation methods revolve around physically-based hydraulic erosion because it can generate realistic natural-looking terrains. However, many of such algorithms were tested only on low resolution terrains. When simulated on a higher resolution terrain, most of the current algorithms become computationally expensive. This is why in many applications today, terrains are generated off-line and loaded during the application runtime. This method restricts the number of terrains which can be stored if there is a limitation on storage capacity.

Recently, graphics hardware has evolved into an indispensable tool in improving the speed of computation. This has motivated us to develop an erosion algorithm to map to graphics hardware for faster terrain generation.

In this paper, we propose a fast and efficient hydraulic erosion procedural technique that utilizes the GPUs powerful computation capability in order to generate high resolution erosion on terrains. Our method is based on the Newtonian physics approach that is implemented on a two-dimensional data structure which stores height fields, water amount, and dissolved sediment and water velocities. We also present a comprehensive comparison between the CPU and GPU implementations together with the visual results and the statistics on simulation time taken.

CR Categories: I.3.5 [Computer Graphics]: Physically based modeling; I.3.6 [Computer Graphics]: Methodology and Techniques; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism;

Keywords: visual simulation, natural phenomena, physically based modeling, terrain, hydraulic erosion.

1 Introduction

Computer graphics researchers recently have made incredible achievements towards developing efficient terrain modelling techniques to generate a synthetic terrain. As graphics hardware becomes more powerful, the demand for more realistic objects such as terrains in computer graphics has increased considerably.

*e-mail: anh@pmail.ntu.edu.sg
†e-mail: assourin@ntu.edu.sg
‡e-mail: parimal@tqglobal.com.sg

Figure 1: *Terrain features with fractional Brownian motion.* [Mandelbrot and Wheeler 1983]

There are two main existing terrain synthesis directions: fractal terrain modeling and physical hydraulic erosion. At first glance, fractal terrains look convincing imitation of natural mountain terrains. Figure 1 shows an example of fractal features in mountain.

However, these fractal terrains have no global erosion features inherently due to isotropy [Musgrave et al. 1989]. Figure 2 is a photo of Fuerteventura Canary Island which was carved into fantastic shapes by the occasional rain. This is an example of eroded terrain which fractal modelling can not represent.

Physically-based hydraulic erosion methods have answered this problem. The methods are not only able to create erosion ridges, canyons and gullies but also can add fractal details to the terrain.

Nevertheless, most of the existing researches are only experimented with small or low resolution terrains. Simulating existing hydraulic erosion method on larger terrains might take very long time to produce good results. That is the reason why most of the terrains are only generated off-line and this provides a little flexibility for the application.

Recently, we have seen fast improvements in graphics hardware performance [Owens et al. 2007]. Coupled with its programmability, graphics hardware has become an ideal platform for computationally demanding tasks in many application domains. This has led us to think of a new mechanism, General-purpose computation on graphics processors (GPGPU), to develop the next generation of GPGPU erosion algorithms for faster terrain generation.

Figure 2: *Fuerteventura Canary Island*.
Source: http://www.canaryventura.com.

2 Related Work

During the past decade, many algorithms have been developed to create efficient models for natural terrains approximation. There are two main approaches to generating a synthetic terrain: fractal landscape modeling and physical erosion simulation.

2.1 Fractal Terrain

The word "fractal" was first introduced more than thirty years ago by a mathematician, Benoit Mandelbrot, father of fractal geometry [Mandelbrot and Wheeler 1983]. Many stochastic subdivision methods have been proposed since then. Fournier et al. [1982] proposed an approach to model both primitives and their motion as a combination of both deterministic and stochastic features. Voss [1985] added successive random displacement to fractional Brownian surfaces. Lewis [1987] developed a generalized stochastic subdivision to deliver random functions with prescribed correlations and spectra. Szeliski and Terzopoulos [1989] presented constrained fractals, a hybrid of splines and fractals that closely blends their complementary features, to solve the problem of user control. Sala et al. [2002] introduced an application of mathematics in a fractal geometry to describe territory and landscape. In 2004, Sharifi et al. [2004] presented a new fractal-based approach for 3D visualization of mountains in VRML standard. Stachniak and Stuerzlinger [2005] presented an algorithm for automated fractal terrain deformation with a set of specified constraints.

2.2 Erosion Terrain

Fractal techniques used to be considered as the most efficient methods for creating realistic-looking terrains. However, fractal terrains in general can only represent rough surfaces and have no global erosion features due to isotropy and stationary [Musgrave et al. 1989]. Lewis [1990] have presented a paper to illustrate fractal modelling is not adequate to describe natural phenomena.

Many promising algorithms introduced later used physically-based approaches to simulate erosion and deposition process and created a realistically better terrain. The first fractal based algorithms for hydraulic erosion was introduced by Kelley et al. [1988]. Kelley showed that greater realism can be achieved from water erosion. The developed model created topographic structure by tracking the “negative space” formed by a pregenerated drainage network. Another technique that also needs a pregenerated river network was published by Nagashima [1998]. The pregenerated river network is created with a two-dimensional fractal function. Then, the river banks are eroded by some physically inspired rules. In 2005, Belhadj and Audibert [2005] introduced a bi-process method which also models realistic terrains with ridge line and drainage network.

As opposed to the methods that generate terrains constrained by a predefined set of drainage network, many other algorithms simulated the erosion process directly on a given terrain model. Prusinkiewicz and Hammel [1993] presented a partial solution incorporating a squig-curve model of a river’s course into the midpoint-displacement model for mountains. This method generates hydraulic eroded terrains by including the generation of rivers in the fractal algorithms. The first physically-based algorithm was described by Musgrave et al. [1989]. Musgrave suggested a global and physical erosion processes which generate both local and global erosion features through a simple simulation. Soil will be dissolved, transported and deposited at lower heights depending on the many conditions, e.g. the altitude difference, water amount and the amount of already-dissolved sediments. This method was extended by Roudier et al. [1993] who took into account possible geological heterogeneities of the ground and the differential erosion that would result. Chiba et al. [1998] introduced another simple but more physically based algorithm for hydraulic erosion which is based on velocity fields of water flows. The simulation obtains the velocity field using a simplified physics formula and simulates the water flow by applying calculated forces caused by the local gradient. In 2002, Benes and Forsbach [2002] presented an improvement of the previous algorithms to simulate hydraulic erosion by distributing all materials to its eight neighbors and introduced an evaporation step to simulate drying pools of water. The authors separated the process into four independent steps which can be applied in arbitrary order to achieve various level of realism. This algorithm and the algorithm in [Musgrave et al. 1989] were suggested further optimizations for a faster method by Olsen [2004] to apply in computer game. For the first time, Olsen proposed a way to define erosion score to measure how eroded a height map is. Olsen proposed many optimization methods but physical exactness and visual appearance are secondary, and applicability in computer game is more important. In 2005, based on layered-data structure [Benes and Forsbach 2001], Neidhold et al. [2005] improved all previous methods and presented a new and much more physically-based erosion method which is claimed to run in real-time for a small resolution terrain. Semi-Lagrangian approximation for fluid called “stable fluid” [Stam 1999] and many physics formulas were simplified to two dimensional version. The paper used Hammersley and Halton points [Wong et al. 1997] to simulate rainfall and provided a detailed method to calculate acceleration, velocity and new positions of water particles based on Newtonian physics approach. In the same year, Benes [2005] proposed another method for visual modelling table mountains without relying on physically correct modelling. Benes [2006] presented another model based on Navier-Stoke equations. These equations form the basis for the model to balance erosion and deposition that determine changes in the layers between water and erosion materials.

However, the presented techniques for hydraulic erosion simulation were demonstrated on small or low resolution terrains. The computationally expensive simulation includes hundreds of steps that can take up to hours to simulate the process in a high resolution or larger terrain.

3 The Proposed Erosion Model

Data structure is a mechanism to save the terrain height data. Many papers on erosion topics have used the common two dimensional regular height field data structure [Musgrave et al. 1989; Roudier et al. 1993; Nagashima 1998; Benes and Forsbach 2002] because of its low memory advantage. However, not every terrain type can be represented by a height field data structure, for example, a terrain with caves.

Dorsey et al. [1999] and Ozawa and Fujishiro [2000] used voxel grid data structure in their modelling and rendering of weathering stones. Although a voxel grid can represent caves, it requires much more storage space.

In our design, the layered data representation [Benes and Forsbach 2001] is selected. Four layers are used to save all related data including the terrain level H, the water amount W, the sediment S and the velocity $\vec{v}$ for the fluid and the erosion simulation. The water acceleration will be calculated when needed.

Table 1: *Data grids used in simulation.*

Terrain Level	H
Water Amount	W
Dissolved Sediment	S
Velocity (3D)	$\vec{v}$

In general, there are five main processes in modelling hydraulic erosion:

- Water is distributed over the terrain.
- Water erodes the terrains and dissolves underlying sediments.
- Water moves to its lower neighbors due to gravity.
- Water deposits the dissolved sediments on lower heights.
- Water evaporates due to heat.

3.1 Water Allocation

The most significant influence for a hydraulic erosion system is the allocation of water. However, running water only can erode a very small amount of erosion each time. Whether it is necessary to distribute rainfall thousands of times or maybe just a few times is one of important questions. Simulating many rainfalls can take very long time and it might become unnecessary. Many papers did not address this issue very clearly. Some papers [Musgrave et al. 1989; Chiba et al. 1998] proposed to model water distribution only at the beginning of the simulation. Olsen [2004] proposed to add water to each cell every simulation step. Neidhold in another paper [2005] proposed to distribute rainfalls many times at random intervals.

In our model, water is added over the terrain only at the beginning of the simulation. There is no concept of rain frequency. The following proposed equation describes the amount of water distributed based on each grid cell's height:

$$W = W_{\min} + W_0 \cdot \frac{H_0}{H_{\max}}$$

Where W_0 is the water amount to be distributed, W_{min} is the water amount to distribute on lowest cells; W_0 is the maximum water amount which can be added more on maximum height cell, and H_0 and H_{max} are current height and maximum height.

3.2 Water Movement

There have been many researches to find approximate solutions for the full Navier-Stokes equations which express realistic fluids animation. As we are creating approximated eroded terrains, these equations are optimized to a two-dimensional version. The result compared to real Navier-Stokes solvers is less physically accurate but much faster and meets very well the needs of our erosion algorithm.

The physics foundation for our approach originated in the paper [Neidhold et al. 2005]. The authors used the Newtonian physics approach for fluid simulation. Although in this paper, the new water position is calculated from its velocity, our experiments show that the simulation in which the amount of water is divided and moved to all lower neighbors gives the best erosion result. Water is still accelerated by the gravity and the velocity is still needed to be calculated, but the velocity will not determine the next water position. Assume that the current cell has an amount of water W, the following equation describing the amount of water will be moved to lower neighboring cell:

$$\Delta W = W . \frac{\Delta h}{\sum(\Delta h)}$$

where $\sum(\Delta h)$ is the total of height difference between current cell and all of its lower neighboring cells. Hence, the amount of water moved to the neighboring cell is proportional to the height difference Δh between the current cell and the neighboring cell.

When the water amount ΔW from one cell is moving to its neighboring cell, the new water amount W_{new} is simply an addition of ΔW and the water amount of destination neighboring cell W_{dest} and the new velocity $\vec{v}_{new}$ will be calculated based on averaging principle:

$$W_{new} = W_{dest} + \Delta W$$

$$\vec{v}_{new} = \frac{W_{dest} . \vec{v}_{dest} + \Delta W . \vec{v}_{\Delta W}}{W_{new}}$$

where $\vec{v}_{dest}$ is the velocity of the destination neighboring cell, respectively and $\vec{v}_{\Delta W}$ is the velocity of ΔW .

The amount of sediment changed will be presented in erosion and deposition part.

3.2.1 Acceleration and Velocity Calculation

Water is moved down from the higher position to lower position by gravity. In order to have physically correct simulation of water movement, acceleration and velocity calculation is essential. In real life, water is first spread over all lower neighbors and accelerated gradually. The first paper in erosion [Musgrave et al. 1989] ignores the calculation of acceleration and velocity. The authors just assumed water would be transported over all lower neighbors based on the height difference. Chiba et al. [1998] proposed new quasi-physically based erosion model later to calculate velocity fields of water flow. In his model, acceleration is supposed to occur in the direction with the maximum angle of inclination. In another paper, Neidhold et al. [2005] calculates acceleration by averaging all obtained direction gradient vectors from lower grid cells.

As in our model, water is divided among the Moore neighborhood (eight neighboring cells); each part of the water will be accelerated with the directional derivatives of its own lower neighboring cell.

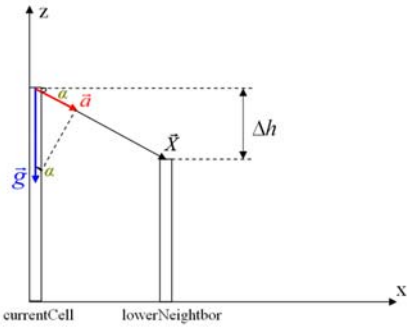


Figure 3: *Acceleration Calculation.*

The formulas to calculate acceleration vector $\vec{a}$ from the gravity vector $\vec{g}$ are as follows:

$$\vec{a} = \vec{g}.\sin\alpha\ ,\ \mathrm{g} \approx 10\ \mathrm{m/s}^2$$

$$\sin\alpha = \frac{\Delta h}{\left|\vec{X}\right|}$$

where $\vec{X}$ is the vector from current cell to its lower neighboring cell. Δh is the height difference between two cells. The height of each cell is considered the total of terrain height and water amount.

The water velocity will be updated according to the acceleration. The following is an optimized equation describing the velocity $\vec{v}$ with respect to its acceleration $\vec{a}$ after the time step Δt at each discrete cell:

$$\vec{v}_{t+\Delta t} = \vec{v}_t.(1 - K_A) + \vec{a}_t.\Delta t$$

where $K_A \in [0..1]$ is the constant to control the sliding friction between the water and terrain.

3.2.2 Evaporation

Evaporation describes the process that the water evaporates and some of the sediments can be deposited to the terrains due to the reduced amount of water. Benes and Forsbach [2002] was the first authors introduced evaporation in erosion model. To the best of our knowledge, no papers mentioned evaporation before that. Many papers proposed later used exponential equation to reduce the water amount by evaporation [Benes and Forsbach 2002; Benes 2005; Neidhold et al. 2005]. However, this exponent function is still computationally costly; so evaporation function W is optimized to a simply discrete version for better performance:

$$W_{t+\Delta t} = W_t.K_E$$

where $K_E \in [0..1]$ is the evaporation constant to control the process.

Neidhold et al. [2005] proposed a varying evaporation based on low temperatures on high terrain levels and high temperatures for a more correct physical evaporation, however the computation is still more expensive and not necessary.

3.3 Erosion and Deposition

Hydraulic erosion functions are used to determine whether to deposit or to dissolve the sediment in each simulation step. As in other erosion models, our erosion algorithm uses some material specific constants which are controlling the entire process:

- Sediment Capacity Constant K_c = 250
- Deposition Constant K_d = 0.05
- Dissolving Constant K_s = 0.04

The Sediment Capacity Constant K_c describes how much sediment can be dissolved in one unit of water at a unit velocity. The deposition constant $K_d \in [0..1]$ controls the rate at which sediment is deposited at the destination cell and the Dissolving Constant $K_S \in [0..1]$ controls the dissolving rate of the underlying terrain into the water after each simulation step.

Whether to deposit or to dissolve the sediment is dependent on the Sediment Capacity S_{cap} which indicates the maximum amount of sediment can be transported by the water.

In general, the sediment capacity S_{cap} is proportional to the Sediment Capacity Constant K_C, transported water amount ΔW and the length of velocity $\vec{v}$:

$$S_{cap} = K_c.\Delta W.\left|\vec{v}\right|$$

This Sediment Capacity S_{cap} will be compared to the actual amount of dissolved sediment ΔS . The following formulas will update data of destination cell:

If $\Delta S > S_{cap}$, then deposit:

$$H' = H + K_d.(\Delta S - S_{cap})$$
$$S' = S + \Delta S - K_d.(\Delta S - S_{cap})$$

Otherwise, dissolve:

$$H' = H - K_s.(S_{cap} - \Delta S)$$
$$S' = S + \Delta S + K_s.(S_{cap} - \Delta S)$$

where H and S are height and sediment of destination cell before the water moves from its neighboring cell while H' and S' are the updated height and sediment of the destination cell. ΔS and S_{cap} are the dissolved sediment and sediment capacity of the transported water, not of the water in destination cell.

4 GPU Implementation

The platform which was used to develop and test the erosion model is a Pentium D 3.2Ghz dual core equipped with NVIDIA GeForce *7900 GS* graphic card. In this project, Visual Studio C++ was used for the erosion implementation on CPU whereas DirectX 9.0 and High Level Shading Language (HLSL) on GPU. This section starts with the data packing in GPU implementation, the memory requirement and then how the erosion algorithm is modified reversely to be implemented in shader code.

4.1 Data Packing

In GPU programming, textures are usually used to store the input data. As we have four layers of data to store in each step, height, fluid, sediment and the 3D velocity, two textures must be used as the input. Table 2 shows in details how these data are arranged. Direct3D data format 32 bit floating point should be used to secure the accuracy.

Table 2: *Erosion Simulation GPU data packing*.

	Red	Green	Blue	Alpha
Texture 1	Height	Fluid	Sediment	Other values
Texture 2	Velocity.x	Velocity.y	Velocity.z	Other values

4.2 Memory Requirement

Memory requirement is one of the major problems in GPGPU programming. But our computation memory requirement is higher than what we have in the graphics card.

The erosion shading program needs two textures as inputs and returns two textures as outputs. Therefore, there are at least four textures required. In order for the simulation to have higher accuracy, 32 bit floating point per channel texture data format should be required. Here is how memory required to store textures is computed:

$$\text{Memory} = \text{Terrain Size} \,.\, \text{num of bit/channel} \,.\, 4 \text{ channels} \,.\, 4 \text{ textures}$$

Table 3 shows that 256 MB video memory is needed to simulate erosion on 2Kx2K terrain if 32 bit floating point per channel data format is used. However, this is the maximum memory amount that the GPU has and therefore 16 bit floating point per channel data format is replaced for 2Kx2K terrain case. Accuracy is slightly reduced but it is still acceptable in the simulation.

Table 3: *GPU Memory Requirement.*

	16 bit floating/channel	32 bit floating/channel
1Kx1K	32 MB	64 MB
2Kx2K	128 MB	256 MB

4.3 Reverse Erosion Algorithm

In Pixel Shader, scattering can not be done but only gathering. In other words, GPU can only write or modify the current pixel value and can not write or modify the pixel's neighbor values (except for CUDA language). Therefore, in order to implement in GPU fragment program, the erosion algorithm has to be implemented reversely.

```
while (step < StepNumber)
{
 if (step = 0) Water Distribution; end if;
 Water Evaporation;
  for each cell in the Height Map
  {
   for each higher neighbor
   {
      Calculate acceleration vector based on height difference
      Calculate fluid velocity vector based on this acceleration
      Move part of the fluid from this neighbor to current cell
      Calculate Sediment Capacity based on the neighbor velocity
        magnitude, fluid amount to decide whether to deposit or dis-
        solve the sediment to current cell
   }
   end for;
   if there is water in the cell and there is lower neighbor
       Remove all water;
   end if;
   if there is no lower neighbor (local minima)
       Deposit part of the sediments;
   end if;
  }
  end for;
}
end while;
```

Algorithm 1*: Reverse erosion algorithm.*

In the erosion algorithm, each current cell is checked whether the cell has water. If yes, the water will be moved to lower neighbors. In the reverse algorithm, the neighbors of current cell are also checked to see if they have any water to move to current cell.

Therefore the computation will be more expensive in this case. As for each current cell, we have to recalculate the amount of water

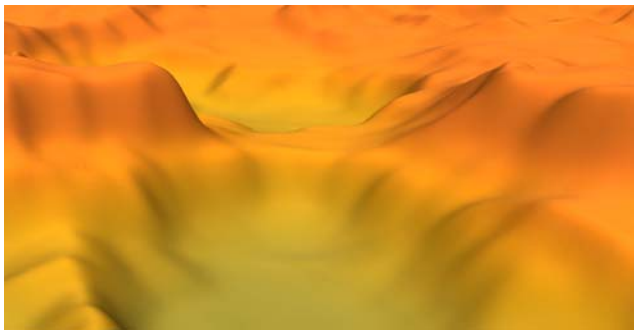
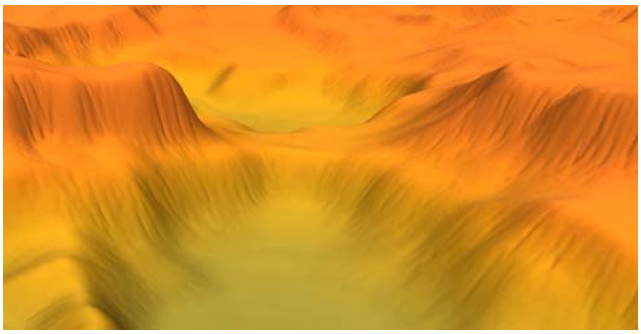

(a)

(b)

Figure 4: *2Kx2K Original (a) and Eroded (b) Terrain.*

to move from the neighbors. Algorithm 1 illustrates the pseudocode of this reverse erosion algorithm.

Our experiments show that if implemented in C++ code only this reverse erosion algorithm takes *5 times* longer than the normal erosion algorithm. However, this is the only way to implement the algorithm in GPU program. With its powerful computation capabilities, it is much faster than CPU normal erosion program.

4.4 GPU shading program

There are many simulation steps in the erosion model. At each step, a shader including three passes will be called. Each pass includes one vertex shader and one pixel shader. Only one vertex shader was used for the three passes and three pixel shaders were written separately. The details of the passes are as follows:

- AddRain Pass: Rain is distributed first in the simulation. In this pass, the fluid amount is simply added to each cell. This job is easily done in C++ code. However, to speed up the whole program, rain distribution was also written in shader. This pass is only called once in the program.
- PreComputation Pass: in the reverse erosion algorithm, the neighbor's total neighbor height difference is calculated many times. PreComputation pass was written to prevent this recalculation in order to speed up the whole program.
- Main Pass: doing main job for the whole erosion model. This main pass reads the two textures from the last simulation steps and uses the erosion algorithm to update and return two new output textures. These new textures are the feedback to next simulation step.

4.5 GPU implementation results

The eroded terrain returned from the GPU implementation is quite similar to CPU implementation result. Figure 4 shows a small part of 2Kx2K original terrain and the terrain after erosion simulation. Figure 5 shows a large view of the terrain.

Performance Statistics

The GPU implementation has proved its powerful computational capabilities. In general, the erosion simulation in GPU can be six to ten times faster than in CPU. For comparison, the following table shows the approximate time of the simulation with 1Kx1K and 2Kx2K terrains in CPU and GPU in 50 and 100 simulation steps:

Table 4: *Time Comparison between CPU and GPU version.*

	1Kx1K Terrain		2Kx2K Terrain	
	CPU	GPU	CPU	GPU
50 steps	67 s	*11.2 s*	290 s	*26.8 s*
100 steps	82 s	*17.2 s*	342 s	*44.3 s*

4.6 Optimization

The faster program achieved from GPGPU is not just from the powerful computational capabilities of the GPU. This program also has been optimized in many ways to maximize its speed:

- Store data in smaller type: 32 bit floating point data format is often used to secure the accuracy. However, the erosion simulation on 2Kx2K terrains would not be possible due to memory limitation. By solving some constrains on 16 bit floating point format, this type can be used to reduce memory requirements which allowed us to simulate the erosion process on 2Kx2K terrains.
- Stop transferring data between GPU and CPU memory: transferring data between CPU and GPU after each simulation step is unnecessary and can make the simulation longer. As we can have up to four render target data, two render target data will be set as inputs and the other two as outputs.
- Move neighbor texture coordinate calculation to Vertex Shader: The erosion algorithm needs to access neighboring values. Pixel Shader can be used to calculate the coordinates to access the texture. Since texture coordinates can be interpolated in the rasterizer between vertex shader and pixel shader, this calculation was moved to Vertex Shader.

5 Conclusion and Future Work

In this work, we have proposed a fast and efficient procedural technique for erosion simulation in GPU. The algorithm for water transportation and hydraulic erosion functions is based on Newtonian physics. The program can simulate large size terrains, i.e. 2Kx2K terrains on a very memory limited graphic card GeForce 7900 GS. Our experiments show that the GPU erosion implementation can run six to ten times faster than the CPU erosion implementation and the eroded terrains returned also look as realistic as the eroded terrains returned from previous algorithms.

The developed erosion algorithm can create ridges, valleys on the mountains slopes and deposit the sediments at the mountain bottoms. As a future work, we will develop additional erosion

feature which deepens and carves additional gullies into the terrain plain to create long erosion channel. We also plan to develop a renderer to view the erosion process at interactive rates on better graphic hardware such as the NVIDIA GeForce 8 series using new GPGPU language Compute Unified Device Architecture (CUDA).

Acknowledgement

The authors would like to thank TQ Global Game Studio for supporting and sponsoring this work. This paper is under the project "Improving Visual Quality of Synthetic Terrain using Weathering Simulation".

References

BELHADJ, F. and AUDIBERT, P. 2005. Modeling Landscapes with Ridges and Rivers: Bottom up Approach. In *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, 447-450.

BENES, B. 2006. Hydraulic Erosion: A Survey.

BENES, B., ARRIAGA. 2005. Table Mountains by Virtual Erosion. In *Proceedings of Eurographics Workshop on Natural Phenomena*, 33-40.

BENES, B. and FORSBACH, R. 2001. Layered Data Representation for Visual Simulation of Terrain Erosion. *Spring Conference on Computer Graphics*, 80-86.

BENES, B. and FORSBACH, R. 2002. Visual Simulation of Hydraulic Erosion. *Journal of WSCG* 1, 79–86.

BENEŠ, B. and FORSBACH, R. 2001. Parallel Implementation of Terrain Erosion Applied to the Surface of Mars. In *Proceedings of the 1st international conference on Computer graphics, virtual reality and visualisation*, 53-57.

BENES, B., TESINSKY, V., HORNYS, J. and BHATIA, S. 2006. Hydraulic Erosion. *Computer Animation and Virtual Worlds 17 (2)*, 99–108.

CHIBA, N., MURAOKA, K. and FUJITA, K. 1998. An Erosion Model Based on Velocity Fields for the Visual Simulation of Mountain Scenery. *The Journal of Visualization and Computer Animation* 9,4, 185-194.

DORSEY, J., EDELMAN, A., JENSEN, H. W., LEGAKIS, J. and PEDERSEN, H. K. 1999. Modeling and Rendering of Weathered Stone. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 225-234.

FOURNIER, A., FUSSELL, D. and CARPENTER, L. 1982. Computer Rendering of Stochastic Models. *Communications of the ACM* 25,6, 371-384.

KELLEY, A. D., MALIN, M. C. and NIELSON, G. M. 1988. Terrain Simulation Using a Model of Stream Erosion. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, 263-268.

LEWIS, J. P. 1987. Generalized Stochastic Subdivision. *ACM Transactions on Graphics* 6,3, 167-190.

LEWIS, J. P. 1990. Is the Fractal Model Appropriate for Terrain? *Disney Secret Lab*.

MANDELBROT, B. B. and WHEELER, J. A. 1983. *The Fractal Geometry of Nature*.

MUSGRAVE, F. K., KOLB, C. E. and MACE, R. S. 1989. The Synthesis and Rendering of Eroded Fractal Terrains. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, 41-50.

NAGASHIMA, K. 1998. Computer Generation of Eroded Valley and Mountain Terrains. *The Visual Computer* 13,9, 456-464.

NEIDHOLD, B., WACKER, M. and DEUSSEN, O. 2005. Interactive Physically Based Fluid and Erosion Simulation. In *Proceedings of Eurographics Workshop on Natural Phenomena*, 25–32.

OLSEN, J. 2004. Realtime Procedural Terrain Generation. *Published Oct*.

OWENS, J. D., LUEBKE, D., GOVINDARAJU, N., HARRIS, M., KRUGER, J., LEFOHN, A. E. and PURCELL, T. J. 2007. A Survey of General-Purpose Computation on Graphics Hardware. *Eurographics 2005, State of the Art Reports*, 21–51.

OZAWA, N. and FUJISHIRO, I. 2000. A Morphological Approach to Volume Synthesis of Weathered Stones. *Volume Graphics workshop'99*, 207-220.

PRUSINKIEWICZ, P. and HAMMEL, M. 1993. A Fractal Model of Mountains with Rivers. *Proceedings of Graphics Interface* 93, 174–180.

ROUDIER, P., PEROCHE, B. and PERRIN, M. 1993. Landscapes Synthesis Achieved through Erosion and Deposition Process Simulation. *Computer Graphics Forum* 12,3, 375-383.

SALA, N., METZELTIN, S. and SALA, M. 2002. Applications of Mathematics in the Real World: Territory and Landscape. *University of Italian Switzerland*.

SHARIFI, M., GOLPAYGANI, F. H. and ESMAELI, M. 2004. A New Fractal-Based Approach for 3d Visualization of Mountains in Vrml Standard. In *Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Austalasia and Southe East Asia*, 100-105.

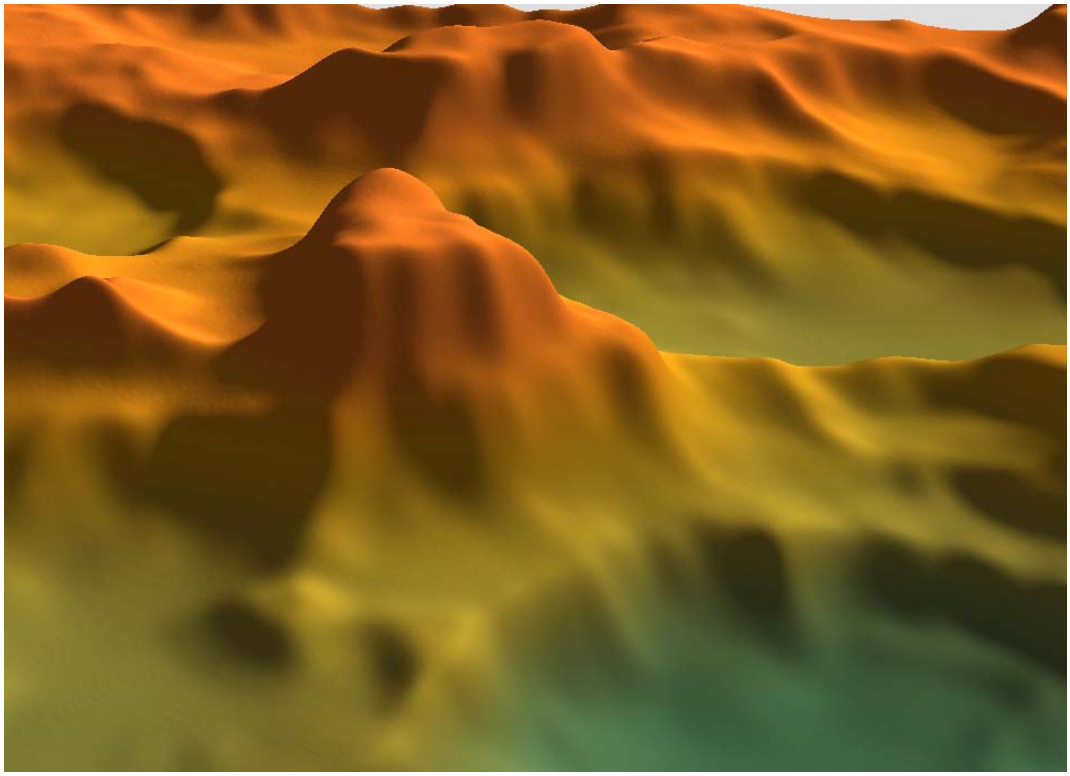
STACHNIAK, S. and STUERZLINGER, W. 2005. An Algorithm for Automated Fractal Terrain Deformation. In *Proceedings of Computer Graphics and Artificial Intelligence*, 64–76.

STAM, J. 1999. Stable Fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 121-128.

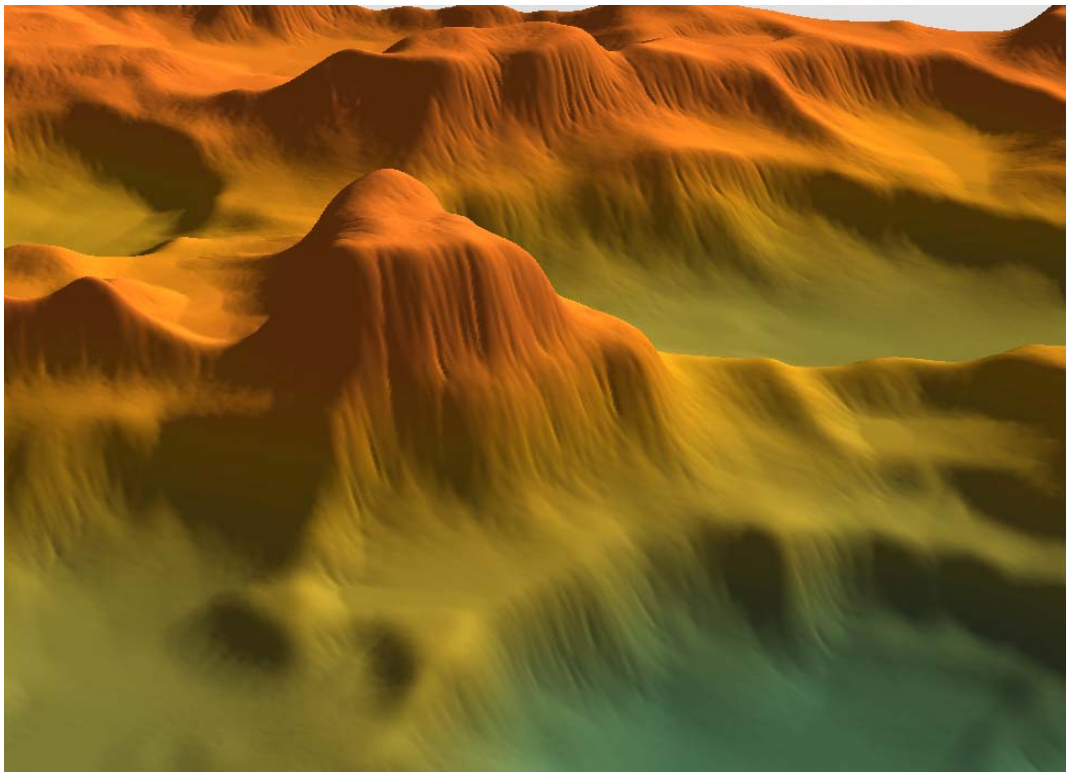
SZELISKI, R. and TERZOPOULOS, D. 1989. From Splines to Fractals. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, 51-60.

VOSS, R. F. 1985. Random Fractal Forgeries. *Fundamental Algorithms for Computer Graphics* 835.

WONG, T. T., LUK, W. S. and HENG, P. A. 1997. Sampling with Hammersley and Halton Points. *Journal of Graphics Tools* 2,2, 9-24.



(a)



(b)

Figure 5: *A Larger View of Original Terrain (a) and Eroded Terrain (b).*